1 Introduction

1.1 About this book

This book is intended to help scientists and engineers learn version 3 of the Python programming language and its associated NumPy, SciPy and Matplotlib libraries. No prior programming experience or scientific knowledge in any particular field is assumed. However, familiarity with some mathematical concepts such as trigonometry, complex numbers and basic calculus is helpful to follow the examples and exercises.

Python is a powerful language with many advanced features and libraries; while the basic syntax of the language is straightforward to learn, it would be impossible to teach it in depth in a book of this size. Therefore, we aim for a balanced, broad introduction to the central features of the language and its important libraries. The text is interspersed with examples relevant to scientific research, and at the end of most sections there are questions (short problems designed to test knowledge) and exercises (longer problems that usually require a short computer program to solve). Although it is not necessary to complete all of the exercises, readers will find it useful to attempt at least some of them. Where a section, example or exercise contains more advanced material that may be skipped on first reading, this is indicated with the symbol ♢.

In Chapter 2 of this book, the basic syntax, data structures and flow control of a Python program are introduced. Chapter 3 is a short interlude on the use of the Pylab library for making graphical plots of data: this is useful to visualize the output of programs in subsequent chapters. Chapter 4 provides more advanced coverage of the core Python language and a brief introduction to object-oriented programming. There follows another short chapter introducing the popular IPython and IPython Notebook environments, before chapters on scientific programming with NumPy, Matplotlib and SciPy. The final chapter covers more general topics in scientific programming, including floating point arithmetic, algorithm stability and programming style.

Readers who are already familiar with the Python programming language may wish to skim Chapters 2 and 4.

Code examples and exercise solutions may be downloaded from the book's website at scipython.com. Note that while comments have been included in these downloadable programs, they are not so extensive in the printed version of this book: instead, the code is explained in the text itself through numbered annotations (such as ❶). Readers typing in these programs for themselves may wish to add their own explanatory comments to the code.

1.2 About Python

Python is a powerful, general-purpose programming language devised by Guido van Rossum in 1989.[1] It is classified as a high-level programming language in that it automatically handles the most fundamental operations (such as memory management) carried out at the processor level ("machine code"). It is considered a higher-level language than, for example, C, because of its expressive syntax (which is close to natural language in some cases) and rich variety of native data structures such as lists, tuples, sets and dictionaries. For example, consider the following Python program which outputs a list of names on separate lines.

Listing 1.1 Outputing a list of names using a program written in Python

```
# eg1-names.py: output three names to the console.

names = ['Isaac Newton', 'Marie Curie', 'Albert Einstein']
for name in names:
    print(name)
```

Output:

```
Isaac Newton
Marie Curie
Albert Einstein
```

Now compare this with the equivalent program in C.

Listing 1.2 Outputing a list of names using a program written in C

```
/* eg1-names.c: output three names to the console. */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_STRING_LENGTH 20
#define NUMBER_OF_STRINGS 3

int main()
{
int i;
char names[NUMBER_OF_STRINGS][MAX_STRING_LENGTH+1];
strcpy(names[0], "Isaac Newton");
strcpy(names[1], "Marie Curie");
strcpy(names[2], "Albert Einstein");

for (i=0;i<NUMBER_OF_STRINGS;i++) {
    fprintf(stdout, "%s\n", names[i]);
}

return EXIT_SUCCESS;
}
```

[1] Python's "benevolent dictator for life."

Even if you are not familiar with the C language, you can see there is quite a lot of overhead involved in coding even this simple task in C: three `includes` of libraries not loaded by default, explicit declarations of variables to hold the list (“array”, in C) of names, `names`, and a counter, `i`, and explicit indexing of this array in a `for` loop; you even need to add the line endings (‘`\n`’ is the “new line” character). This source code then has to be *compiled* – converted into the machine code that the computer processor understands – before it can be run (*executed*). Furthermore, there is plenty of scope for errors (bugs): trying to print the name stored in `name[10]` will likely cause junk to be output: the C compiler won’t stop you from accessing this nonexistent name.

The same program written in three lines of Python is clean and expressive: we do not have to explicitly declare that `names` is a list of strings, there is no need for a loop counter like `i` and there are no separate libraries to include (`import` in Python). To run the Python program, one simply needs to type `python eg1-names.py` which will automatically invoke the Python “interpreter” to compile and then run the resulting “byte-code” (a kind of intermediate representation of the program between its source and the ultimate machine code that Python dispatches to the processor).

Python’s syntax aims to ensure that “There should be one – and preferably only one – obvious way to do it.” This differs from some other popular high-level languages such as Ruby and Perl, which take the opposite approach, encapsulated by the mantra “there’s more than one way to do it.” For example, there are (at least) four obvious ways to output the same list in Perl:[2]

Listing 1.3 Different ways to output a list of names using a program written in Perl

```
@names = ("Isaac Newton", "Marie Curie", "Albert Einstein");
# Method 1
print "$_\n" for @names;

# Method 2
print join "\n", @names;
print "\n";

# Method 3
print map { "$_\n" } @names;

# Method 4
$" = "\n";
print "@names\n";
```

(Note also Perl’s famously concise but somewhat opaque syntax.)

1.2.1 Advantages and disadvantages of Python

Here are some of the main advantages of the Python programming language and why you might want to use it:

[2] Well, obvious to Perl programmers.

- Its clean and simple syntax makes writing Python programs fast and generally minimizes opportunities for bugs to creep in. When done right, the result is high-quality software that is easy to maintain and extend.
- It's free – Python and its associated libraries are free of cost and open source, unlike commercial offerings such as Mathematica.
- Cross-platform support: Python is available for every commonly available computer system, including Windows, Unix, Linux and Mac OS X. Although platform-specific extensions exist, it is possible to write code that will run on any platform without modification.
- Python has a large library of modules and packages that extend its functionality. Many of these are available as part of the "standard library" provided with the Python interpreter itself. Others, including the NumPy, SciPy and Matplotlib libraries used in scientific computing, can be downloaded separately for no cost.
- Python is relatively easy to learn. The syntax and idioms used for basic operations are applied consistently in more advanced usage of the language. Error messages are generally meaningful assessments of what went wrong rather than the generic "crashes" that can occur in compiled lower-level languages such as C.
- Python is flexible: it is often described as a "multi-paradigm" language that contains the best features from the procedural, object-oriented and functional programming paradigms. There is little need for the work-arounds required in some languages when a problem can only be solved cleanly with one of these approaches.

So where's the catch? Well, Python does have some disadvantages and isn't suitable for every application.

- The speed of execution of a Python program is not as fast as some other, fully compiled languages such as C and Fortran. For heavily numerical work, the NumPy and SciPy libraries alleviate this to some extent by using compiled-C code "under the hood," but at the expense of some reduced flexibility. For many, many applications, however, the speed difference is not noticeable and the reduced speed of execution more than offset by a much faster speed of *development*. That is, it takes much less time to write and debug a Python program than to do the same in C, C++ or Java.
- It is hard to hide or obfuscate the source code of a Python program to prevent others from copying or modifying it. However, this doesn't mean that successful commercial Python programs don't exist.
- A common complaint about Python has historically been that its rapid development has led to compatibility issues between versions. Certainly there are important differences between Python 2 and Python 3 (described in the next section), but the complaint stems from the fact that within the Python 2 series, there were major improvements and additions to the language that meant that code written in a later version (say, 2.7) would not run on an earlier version of Python (e.g., 2.6), although code written for an earlier version of Python will always run on a later version (within the same branch, 2 or 3). If you use the

latest version of Python (see Section 1.3) you probably won't run into a problem, but some operating systems that come with Python are rather conservative and install by default only an older version.

1.2.2 Python 2 or Python 3?

At the time of writing, Python users have a choice to make: whether to use the older, more established Python 2 version of the language or the newer Python 3. Although the differences between the two versions may seem minor, code written in Python 3 will not run under Python 2 and vice versa: Python 3 is not *backward-compatible* with its predecessor. **This book teaches Python 3**.

The latest major version of Python 2, Python 2.7, will be the last of that branch. Since its release in 2009, the number of users and extent of library support for Python 3 has grown to the point that new users would find little benefit in learning Python 2 except to maintain legacy code.

There are several reasons for major change between versions (breaking your users' existing code is not something to be undertaken lightly): Python 3 fixes some ugly quirks and inconsistencies in the language and provides *Unicode support* for all strings (eliminating a lot of the confusion that is created in dealing with Unicode and non-Unicode strings in Python 2). Unicode is an international standard for the representation of text in most of the writing systems in the world.

It is anticipated that most users of this book will not have trouble converting their own code between the two versions of Python if necessary. Where important, the differences are pointed out in the text.

1.3 Installing Python

The official website of Python is www.python.org/, and contains full and easy-to-follow instructions for downloading Python. However, there are several full distributions which include the NumPy, SciPy and Matplotlib libraries (the "SciPy stack") to save you from having to download and install these yourself:

- *Anaconda* is available for free (including for commerical use) from http://continuum.io/downloads. It installs both Python 2 and Python 3, but the default version can be selected either before downloading as indicated on this web page, or subsequently using the '`conda`' command.
- *Enthought Canopy* is a similar distribution with a free version and various tiers of paid-for versions including technical support and development software.

In most cases, one of these distributions should be all you need. We provide some platform-specific notes below.

The source code (and binaries for some platforms) for the NumPy, SciPy, Matplotlib and IPython packages are available separately at:

- NumPy: http://sourceforge.net/projects/numpy/
- SciPy: http://sourceforge.net/projects/scipy/
- Matplotlib: http://matplotlib.org/downloads.html
- IPython: https://github.com/ipython/ipython/releases

Windows

Windows users have a couple of further options for installing the full SciPy stack: *Python(x,y)* (https://code.google.com/p/pythonxy/) and *WinPython* (http://winpython.sourceforge.net/). Both are free.

Mac OS X

Mac OS X, being based on Unix, comes with Python, but it is usually an older version of Python 2. You must not delete or modify this installation (it's needed by the operating system), but you can follow the instructions above for obtaining Python 3 and the SciPy stack. OS X does not have a native *package manager* (an application for managing and installing software), but the two popular third-party package managers, Homebrew (http://brew.sh/) and MacPorts (www.macports.org/), can both supply Python 3 and its packages if you prefer this option.

Linux

Almost all Linux distributions come with Python 2, but usually not Python 3, so you will need to install it from the links above: the Anaconda and Canopy distributions both have versions for Linux. Most Linux distributions come with their own software package managers (e.g., `apt` in Debian and `rpm` for RedHat). These can be used to install Python 3 and its libraries, though finding the necessary package repositories may take some research on the Internet. Be careful not to replace or modify your system installation as other applications may depend on it.

1.4 The command line

Most of the code examples in this book are written as standalone programs which can be run from the *command line* (or from within an *integrated development environment* (IDE) if you use one: see Section 9.3.2). To access the command line interface (also known as a console or terminal) on different platforms, follow the instructions below.

- Windows 7 and earlier: *Start* > *All Programs* > *Command Prompt*; alternatively, type `cmd` in the *Start* > *Run* input box.
- Windows 8: *Preview* (lower left of screen) > *Windows System: All apps*; alternatively type '`cmd`' in the search box pulled down the top-right corner of the screen.
- Mac OS X: *Finder* > *Applications* > *Utilities* > *Terminal*
- Linux: if you are not using a graphical interface you are already at the command line; if you are, then locate the Terminal application (distributions vary, but it is usually found within a *System Utilities* or *System Tools* subfolder).

Commands typed at the command line are interpreted by an application called a *shell*, which allows the user to navigate the file system and is able to start other applications. For example, the command

```
python myprog.py
```

instructs the shell to invoke the Python interpreter, sending it the file `myprog.py` as the script to execute. Output from the program is then returned to the shell and displayed in your console.